Basic

Evaluation Criteria

- Scribe - 5%
- Weekly Quizzes - 20%
- Mid term 1 - 15%
- Mid term 2 - 15%
- Final Exam - 45%

Protocol

A protocol defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message or other event.

Packet Switching

- End systems exchange messages with each other.
- Long messages into smaller chunks of data known as packets.
- Between source and destination, each packet travels through communication links and packet switches
- Packets are transmitted over each communication link at a rate equal to the full transmission rate of the link. So, if a source end system or a packet switch is sending a packet of L bits over a link with transmission rate R bits/sec, then the time to transmit the packet is L/R seconds.
- Each packet can transfer via different path.

Store And Forward

Store-and-forward transmission means that the packet switch (router) must receive the entire packet before it can begin to transmit the first bit of the packet onto the outbound link (Maybe because we need to process header). Router can store multiple packets.

Question: Consider a simple network consisting of two end systems connected by a single router, as shown in the figure.

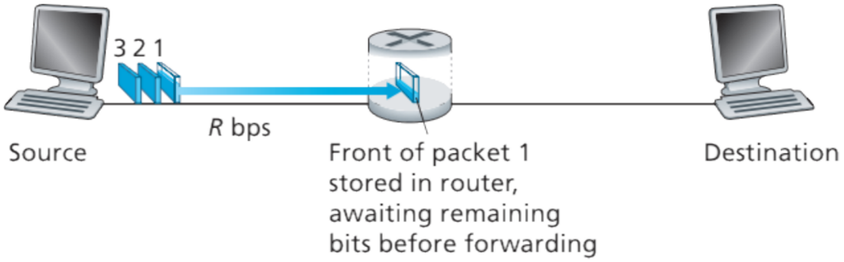

3 2 1
R bps
Source
Front of packet 1 stored in router, awaiting remaining bits before forwarding
Destination

Figure: [Kurose and Ross] Store-and-forward packet switching.

All links in the above figure have transmission rate R bits/sec. The source wants to send 4 packets of L bits each to the destination. If the source starts transmission of the first packet at time 0, at what time would the destination receive all the packets?

Answer: $2L/R + 3L/R = 5L/R$.

Delay in Packet-Switched Networks

- **Processing Delay:** The time required to examine the packets header and determine where to direct the packet is part of the processing delay. The processing delay can also include other factors, such as the time needed to check for bit-level errors in the packet that occurred in transmitting the packets bits from the upstream node to router A. Processing delay in high-speed routers are typically on the order of microseconds or less. After this nodal processing, the router directs the packet to the queue that precedes the link to router B.
- **Queuing Delay:** At the queue, the packet experiences a queuing delay as it waits to be transmitted onto the link. The length of the queuing delay of a specific packet will depend on the number of earlier-arriving packets that are queued and waiting for transmission onto the link. If the queue is empty and no other packet is currently being transmitted, then our packets queuing delay will be zero. On the other hand, if the traffic is heavy and many other packets are also waiting to be transmitted, the queuing delay will be long.Queuing delays can be on the order of microseconds to milliseconds in practice.
- **Transmission Delay:** Assuming that packets are transmitted in a first-come-first-served manner, as is common in packet-switched networks, our packet can be transmitted only after all the packets that have arrived before it have been transmitted. Denote the length of the packet by L bits, and denote the transmission rate of the link from router A to router B by R bits/sec. The transmission delay is L/R. This is the amount of

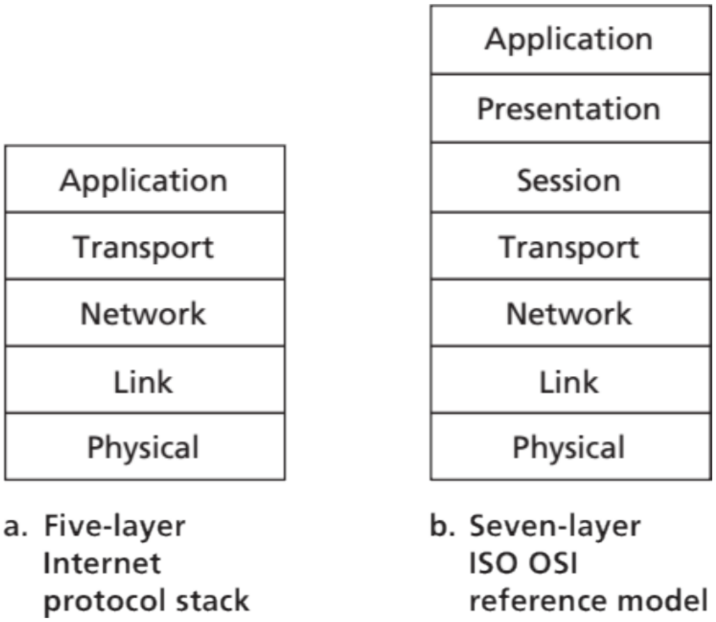
time required to push (that is, transmit) all of the packets bits into the link.

- **Propagation Delay:** Once a bit is pushed into the link, it needs to propagate to router B. The time required to propagate from the beginning of the link to router B is the propagation delay. The bit propagates at the propagation speed of the link. The propagation speed depends on the physical medium of the link. The propagation delay is the distance between two routers divided by the propagation speed. That is, the propagation delay is d/s, where d is the distance between router A and router B and s is the propagation speed of the link. Once the last bit of the packet propagates to node B, it and all the preceding bits of the packet are stored in router B. The whole process then continues with router B now performing the forwarding.

Circuit Switching

- Resources needed along a path (buffers, link transmission rate) to provide for communication between the end systems are reserved for the duration of the communication session between the end systems.
- A circuit in a link is implemented with either frequency-division multiplexing (FDM) or time-division multiplexing (TDM).
- In **FDM**, the frequency spectrum of a link is divided up among the connections established across the link. Specifically, the link dedicates a frequency band to each connection for the duration of the connection.
- **TDM** is considered to be a digital procedure which can be employed when the transmission medium data rate quantity is higher than the data rate requisite of the transmitting and receiving devices. In TDM, corresponding frames carry data to be transmitted from the different sources. Each frame consists of a set of time slots, and portions of each source is assigned a time slot per frame.
- Circuit Switching is basically used for real time applications
- Packet switching is better than circuit switching as even traditional applications of circuit switching such as phone calls is now being done with the help of packet switching which even allows for both of the things, i.e. internet and phone call to happen simultaneously (VoLTE). Also it can easily accommodate more sporadic users without change in configuration.

Protocol Layering



a. Five-layer Internet protocol stack

b. Seven-layer ISO OSI reference model

Figure: [Kurose and Ross] The Internet protocol stack (a) and OSI (Open Systems Interconnection) reference model (b).

Layer interacts with layers only next to it.

Benefits of layering: * We can modify a layer freely, i.e. tweak it. * Layer acts as a wrapper which can be unwrapped at the other end.

Application Layer

The application layer is where network applications and their application-layer protocols reside. (Like HTTP, FTP, DNS, etc) An application-layer protocol is distributed over multiple end systems, with the application in one end system using the protocol to exchange packets of information with the application in another end system. We'll refer to this packet of information at the application layer as a message.

Transport Layer

The Internet's transport layer transports application-layer messages between application endpoints. In the Internet there are two transport protocols, TCP and UDP, either of which can transport application-layer messages. TCP provides a connection-oriented service to its applications. This service includes guaranteed delivery of application-layer messages to the destination and flow control (that is, sender/receiver speed matching). TCP also breaks long messages into shorter segments and provides a congestion-control mechanism, so that a source throttles its transmission rate when the network is congested. The UDP protocol provides a connectionless service to its applications. This is a no-frills service that provides no reliability, no flow control, and no congestion control. We'll refer to a transport-layer packet as a segment.

Network Layer

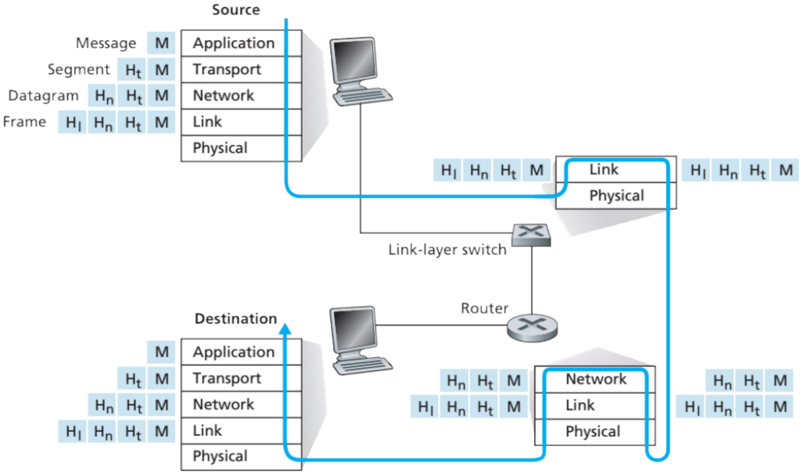
The Internet's network layer is responsible for moving network-layer packets known as datagrams from one host to another. The Internet's network layer includes the celebrated IP Protocol, which defines the fields in the datagram as well as how the end systems and routers act on these fields.

Link Layer

The Internet's network layer routes a datagram through a series of routers between the source and destination. To move a packet from one node (host or router) to the next node in the route, the network layer relies on the services of the link layer. In particular, at each node, the network layer passes the datagram down to the link layer, which delivers the datagram to the next node along the route. At this next node, the link layer passes the datagram up to the network layer. Examples of linklayer protocols include Ethernet, WiFi. As datagrams typically need to traverse several links to travel from source to destination, a datagram may be handled by different link-layer protocols at different links along its route. We'll refer to the linklayer packets as frames.

Physical Layer

While the job of the link layer is to move entire frames from one network element to an adjacent network element, the job of the physical layer is to move the individual bits within the frame from one node to the next. The protocols in this layer are again link dependent and further depend on the actual transmission medium of the link (for example, twisted-pair copper wire, single-mode fiber optics).



[Kurose and Ross] Hosts, routers, and link-layer switches; each contains a different set of layers, reflecting their differences in functionality.

Thus, we see that at each layer, a packet has two types of fields: header fields and a payload field. The payload is typically a packet from the layer above.

Physical Layer

Fourier Analysis

Fourier proved that any reasonably behaved periodic function, $g(t)$ with period T, can be constructed as the sum of a (possibly infinite) number of sines and cosines:

$$g(t) = \frac{1}{2}c + \sum_{n=1}^{\infty} a_n \sin(2\pi n f t) + \sum_{n=1}^{\infty} b_n \cos(2\pi n f t)$$

where $f = 1/T$ is the fundamental frequency, a_n and b_n are the sine and cosine amplitudes of the n th harmonics (terms), and c is a constant.

The a_n amplitudes can be computed for any given $g(t)$ by multiplying both sides of Eq. by $\sin(2\pi kft)$ and then integrating from 0 to T. since

$\int_0^T \sin(2\pi kft)\sin(2\pi nft)dt = \begin{cases} 0 \text{ for } k \neq n \\ T/2 \text{ for } k = n \end{cases}$

only one term of the summation survives: a_n. The b_n summation vanishes completely. Similarly, by multiplying Eq. by $\cos(2\pi kft)$ and integrating between 0 and T, we can derive b_n. By just integrating both sides of the equation as it stands, we can find c. The results of performing these operations are as follows:

$a_n = \frac{2}{T}\int_0^T g(t)\sin(2\pi nft)dt \quad b_n = \frac{2}{T}\int_0^T g(t)\cos(2\pi nft)dt \quad c = \frac{2}{T}\int_0^T g(t)dt$

Bandwidth-Limited Signals

The relevance of all of this to data communication is that real channels affect different frequency signals differently. Let us consider a specific example: the transmission of the ASCII character “b” encoded in an 8-bit byte. The bit pattern that is to be transmitted is 01100010. The left-hand part of Fig. (a) shows the voltage output by the transmitting computer. The Fourier analysis of this signal yields the coefficients:

$$\begin{aligned}
a_n &= \frac{1}{\pi n}[\cos(\pi n/4) - \cos(3\pi n/4) + \cos(6\pi n/4) - \cos(7\pi n/4)] \\
b_n &= \frac{1}{\pi n}[\sin(3\pi n/4) - \sin(\pi n/4) + \sin(7\pi n/4) - \sin(6\pi n/4)] \\
c &= 3/4
\end{aligned}$$

The root-mean-square amplitudes, $\sqrt{a_n^2 + b_n^2}$, for the first few terms are shown on the right-hand side of Fig. 2-1(a). These values are of interest because their squares are proportional to the energy transmitted at the corresponding frequency. No transmission facility can transmit signals without losing some power in the process. If all the Fourier components were equally diminished, the resulting signal would be reduced in amplitude but not distorted [i.e., it would have the same nice squared-off shape as Fig. 2-1(a)]. Unfortunately, all transmission facilities diminish different Fourier components by different amounts, thus introducing distortion. Usually, for a wire, the amplitudes are transmitted mostly undiminished from 0 up to some frequency f_c [measured in cycles/sec or Hertz (Hz)], with all frequencies above this cutoff frequency attenuated. The width of the frequency range transmitted without being strongly attenuated is called the bandwidth. In practice, the cutoff is not really sharp, so often the quoted bandwidth is from 0 to the frequency at which the received power has fallen by half.

The bandwidth is a physical property of the transmission medium that depends on, for example, the construction, thickness, and length of a wire or fiber. Filters are often used to further limit the bandwidth of a signal. 802.11 wireless channels are allowed to use up to roughly 20 MHz, for example, so 802.11 radios filter the signal bandwidth to this size. As another example, traditional (analog) television channels occupy 6 MHz each, on a wire or over the air. This filtering lets more signals share a given region of spectrum, which improves the overall efficiency of the system. It means that the frequency range for some signals will not start at zero, but this does not matter. The bandwidth is still the width of the band of frequencies that are passed, and the information that can be carried de- pends only on this width and not on the starting and ending frequencies. Signals that run from 0 up to a maximum frequency are called baseband signals. Signals that are shifted to occupy a higher range of frequencies, as is the case for all wireless transmissions, are called passband signals. So bandwidth = baseband + passband.

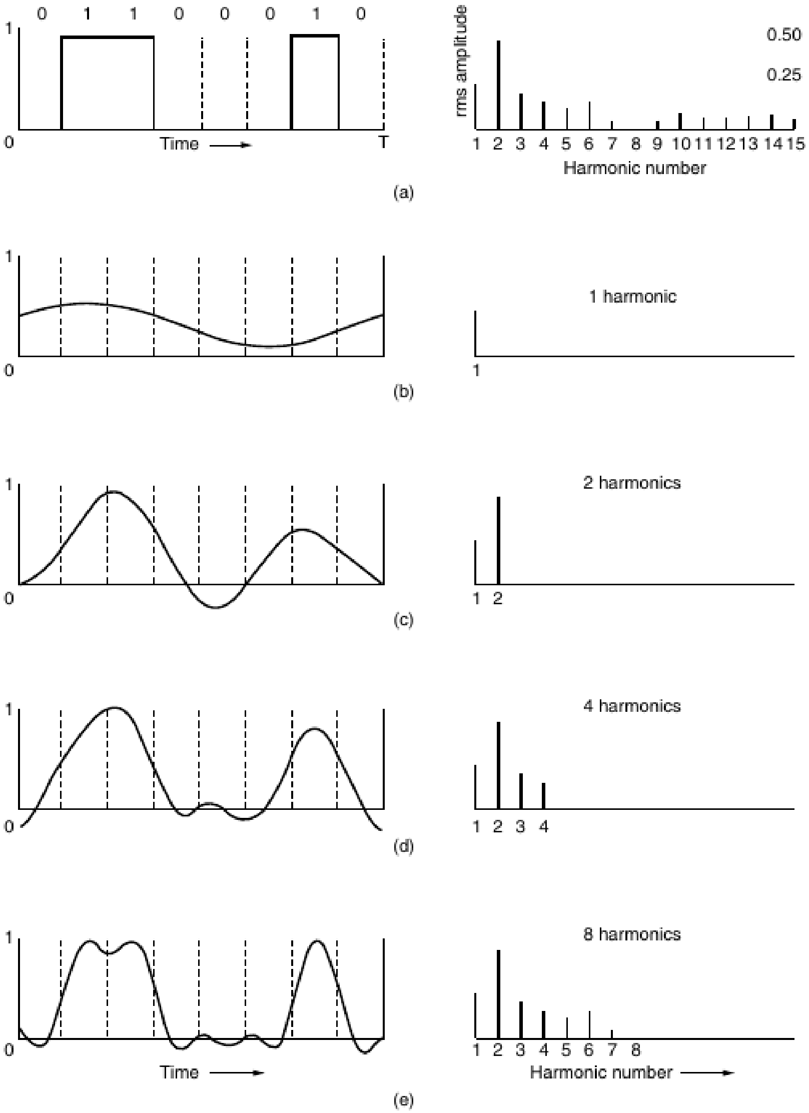


Figure 2-1. [Tanenbaum] (a) A binary signal and its root-mean-square Fourier amplitudes. (b)–(e) Successive approximations to the original signal.

Now let us consider how the signal of Fig. 2-1(a) would look if the bandwidth were so low that only the lowest frequencies were transmitted [i.e., if the function were being approximated by the first few terms of Eq. (2-1)]. Figure 2-1(b) shows the signal that results from a channel that allows only the first harmonic (the fundamental, f) to pass through. Similarly, Fig. 2-1(c)–(e) show the spectra and reconstructed functions for higher-bandwidth channels. For digital transmission, the goal is to receive a signal with just enough fidelity to reconstruct the sequence of bits that was sent. We can already do this easily in Fig. 2-1(e), so it is wasteful to use more harmonics to receive a more accurate replica.

Given a bit rate of b bits/sec, the time required to send the 8 bits in our example 1 bit at a time is 8/b sec, so the frequency of the first harmonic of this signal is b /8 Hz. An ordinary telephone line, often called a voice-grade line, has an artificially introduced cutoff frequency just above 3000 Hz. The presence of this restriction means that the number of the highest harmonic passed through is roughly 3000/(b/8), or 24,000/b (the cutoff is not sharp). For some data rates, the numbers work out as shown in Fig. 2-2. From these numbers, it is clear that trying to send at 9600 bps over a voice-grade telephone line will transform Fig. 2-1(a) into something looking like Fig. 2-1(c), making accurate reception of the original binary bit stream tricky. It should be obvious that at data rates much higher than 38.4 kbps, there is no hope at all for binary signals, even if the transmission facility is completely noiseless. In other words, limiting the bandwidth limits the data rate, even for perfect channels. However, coding schemes that make use of several voltage levels do exist and can achieve higher data rates. We will discuss these later in this chapter.

Bps (b)	T (msec) (8/b)	First Harmonic (Hz) (b/8)	# Harmonics sent
300	26.67	37.5	80
600	13.33	75	40
1200	6.67	150	20
2400	3.33	300	10
4800	1.67	600	5
9600	0.83	1200	2
19200	0.42	2400	1
38400	0.21	4800	0

Figure 2-2. Relation between data rate and harmonics for our example.

The Maximum Data Rate of a Channel

Nyquist proved that if an arbitrary signal has been run through a low-pass filter of bandwidth B, the filtered signal can be completely reconstructed by making only 2B (exact) samples per second. Sampling the line faster than 2B times per second is pointless because the higher-frequency components that such sampling could recover have already been filtered out. If the signal consists of V discrete levels, Nyquist's theorem states: maximum data rate = $2Blog_2V$ bits/sec (2-2) For example, a noiseless 3-kHz channel cannot transmit binary (i.e., two-level) signals at a rate exceeding 6000 bps. So far we have considered only noiseless channels. If random noise is present, the situation deteriorates rapidly. And there is always random (thermal) noise present due to the motion of the molecules in the system. The amount of thermal noise present is measured by the ratio of the signal power to the noise power, called the SNR (Signal-to-Noise Ratio). If we denote the signal power by S and the noise power by N, the signal-to-noise ratio is S/N. Usually, the ratio is expressed on a log scale as the quantity $10\log_{10} S/N$ because it can vary over a tremendous range. The units of this log scale are called decibels (dB), with "deci" meaning 10 and "bel" chosen to honor Alexander Graham Bell, who invented the telephone. An S /N ratio of 10 is 10 dB, a ratio of 100 is 20 dB, a ratio of 1000 is 30 dB, and so on. The manufacturers of stereo amplifiers often characterize the bandwidth (frequency range) over which their products are linear by giving the 3dB frequency on each end. These are the points at which the amplification factor has been approximately halved (because $10\log_{10} 0.5 \approx -3$). Shannon's major result is that the maximum data rate or capacity of a noisy channel whose bandwidth is B Hz and whose signal-to-noise ratio is S/N, is given by: maximum number of bits/sec $= B\log_2(1 + S/N)$ (2-3)

This tells us the best capacities that real channels can have. For example, ADSL (Asymmetric Digital Subscriber Line), which provides Internet access over normal telephone lines, uses a bandwidth of around 1 MHz. The SNR depends strongly on the distance of the home from the telephone exchange, and an SNR of around 40 dB for short lines of 1 to 2 km is very good. With these characteristics, the channel can never transmit much more than 13 Mbps ($4log_2(10) \approx 13$), no matter how many or how few signal levels are used and no matter how often or how infrequently samples are taken. In practice, ADSL is specified up to 12 Mbps, though users often see lower rates. This data rate is actually very good, with over 60 years of communications techniques having greatly reduced the gap between the Shannon capacity and the capacity of real systems.

Digital Modulation And Multiplexing

Wires and wireless channels carry analog signals such as continuously varying voltage, light intensity, or sound intensity. To send digital information, we must devise analog signals to represent bits. The process of converting between bits and signals that represent them is called digital modulation.

We will start with schemes that directly convert bits into a signal. These schemes result in baseband transmission, in which the signal occupies frequencies from zero up to a maximum that depends on the signaling rate. It is common for wires. Then we will consider schemes that regulate the amplitude, phase, or frequency of a carrier signal to convey bits. These schemes result in passband transmission, in which the signal occupies a band of frequencies around the frequency of the carrier signal. It is common for wireless and optical channels for which the signals must reside in a given frequency band. Channels are often shared by multiple signals. After all, it is much more convenient to use a single wire to carry several signals than to install a wire for every signal. This kind of sharing is called multiplexing. It can be accomplished in several different ways. We will present methods for time, frequency, and code division multiplexing.

Baseband Transmission

The most straightforward form of digital modulation is to use a positive voltage to represent a 1 and a negative voltage to represent a 0. This scheme is called NRZ (Non-Return-to-Zero). An example is shown in Fig. 2-20(b).

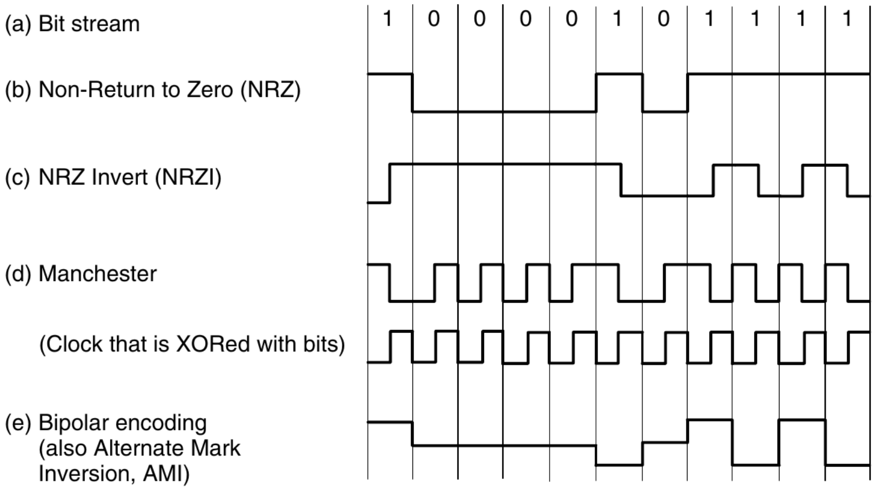

(a) Bit stream
1 0 0 0 0 1 0 1 1 1 1
(b) Non-Return to Zero (NRZ)
(c) NRZ Invert (NRZI)
(d) Manchester
(Clock that is XORed with bits)
(e) Bipolar encoding (also Alternate Mark Inversion, AMI)

Figure 2-3. [Tanenbaum] Line codes: (a) Bits, (b) NRZ, (c) NRZI, (d) Manchester, (e) Bipolar or AMI.

Once sent, the NRZ signal propagates down the wire. At the other end, the receiver converts it into bits by sampling the signal at regular intervals of time.

This signal will not look exactly like the signal that was sent. It will be attenuated and distorted by the channel and noise at the receiver. To decode the bits, the receiver maps the signal samples to the closest symbols. For NRZ, a positive voltage will be taken to indicate that a 1 was sent and a negative voltage will be taken to indicate that a 0 was sent.

One strategy for using limited bandwidth more efficiently is to use more than two signaling levels. By using four voltages, for instance, we can send 2 bits at once as a single symbol. This design will work as long as the signal at the receiver is sufficiently strong to distinguish the four levels. The rate at which the signal changes is then half the bit rate, so the needed bandwidth has been reduced.

We call the rate at which the signal changes the symbol rate to distinguish it from the bit rate. The bit rate is the symbol rate multiplied by the number of bits per symbol.

Clock Recovery For all schemes that encode bits into symbols, the receiver must know when one symbol ends and the next symbol begins to correctly decode the bits. With NRZ, in which the symbols are simply voltage levels, a long run of 0s or 1s leaves the signal unchanged. After a while it is hard to tell the bits apart, as 15 zeros look much like 16 zeros unless you have a very accurate clock.

Accurate clocks would help with this problem, but they are an expensive solution for commodity equipment.

One strategy is to send a separate clock signal to the receiver. Another clock line is no big deal for computer buses or short cables in which there are many lines in parallel, but it is wasteful for most network links since if we had another line to send a signal we could use it to send data. A clever trick here is to mix the clock signal with the data signal by XORing them together so that no extra line is needed. The results are shown in Fig. 2-2(d). The clock makes a clock transition in every bit time, so it runs at twice the bit rate. This scheme is called Manchester encoding. The downside of Manchester encoding is that it requires twice as much band-width as NRZ because of the clock. A different strategy is based on the idea that we should code the data to ensure that there are enough transitions in the signal. Consider that NRZ will have clock recovery problems only for long runs of 0s and 1s. If there are frequent transitions, it will be easy for the receiver to stay synchronized with the incoming stream of symbols.

As a step in the right direction, we can simplify the situation by coding a 1 as a transition and a 0 as no transition, or vice versa. This coding is called NRZI (Non-Return-to-Zero Inverted), a twist on NRZ. An example is shown in Fig. 2-2(c). With it, long runs of 1s do not cause a problem. Of course, long runs of 0s still cause a problem that we must fix. To really fix the problem we can break up runs of 0s by mapping small groups of bits to be transmitted so that groups with successive 0s are mapped to slightly longer patterns that do not have too many consecutive 0s. A well-known code to do this is called 4B/5B. Every 4 bits is mapped into a5-bit pattern with a fixed translation table. The five bit patterns are chosen so that there will never be a run of more than three consecutive 0s. This scheme adds 25% overhead, which is better than the 100% overhead of Manchester encoding. Since there are 16 input combinations and 32 output combinations, some of the output combinations are not used. Putting aside the combinations with too many successive 0s, there are still some codes left. As a bonus, we can use these nondata codes to represent physical layer control signals. For example, in some uses “11111” represents an idle line and “11000” represents the start of a frame.

Data (4B)	Codeword (5B)	Data (4B)	Codeword (5B)
0000	11110	1000	10010
0001	01001	1001	10011
0010	10100	1010	10110
0011	10101	1011	10111
0100	01010	1100	11010
0101	01011	1101	11011
0110	01110	1110	11100
0111	01111	1111	11101

Figure 2-4. [Tanenbaum] 4B/5B mapping.

FDM

FDM (Frequency Division Multiplexing) takes advantage of passband transmission to share a channel. It divides the spectrum into frequency bands, with each user having exclusive possession of some band in which to send their signal.

In Fig. 2-25 we show three voice-grade telephone channels multiplexed using FDM. Filters limit the usable bandwidth to about 3100 Hz per voice-grade channel. When many channels are multiplexed together, 4000 Hz is allocated per channel. The excess is called a guard band. It keeps the channels well separated. First the voice channels are raised in frequency, each by a different amount. Then they can be combined because no two channels now occupy the same portion of the spectrum. Notice that even though there are gaps between the channels thanks to the guard bands, there is some overlap between adjacent channels. The overlap is there because real filters do not have ideal sharp edges. This means that a strong spike at the edge of one channel will be felt in the adjacent one as nonthermal noise.

This scheme has been used to multiplex calls in the telephone system for many years, but multiplexing in time is now preferred instead. However, FDM continues to be used in telephone networks, as well as cellular, terrestrial wireless, and satellite networks at a higher level of granularity.

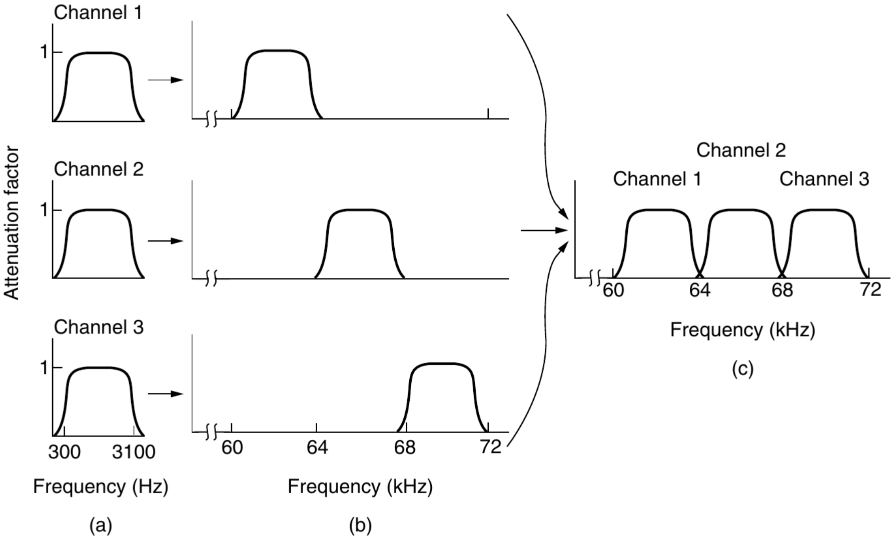


Figure 2-5: [Tanenbaum] Frequency division multiplexing. (a) The original bandwidths. (b) The bandwidths raised in frequency. (c) The multiplexed channel.

TDM

Here, the users take turns (in a round-robin fashion), each one periodically getting the entire bandwidth for a little burst of time. An example of three streams being multiplexed with TDM is shown in Fig. 2-6. Bits from each input stream are taken in a fixed time slot and output to the aggregate stream. This stream runs at the sum rate of the individual streams. For this to work, the streams must be synchronized in time. Small intervals of guard time analogous to a frequency guard band may be added to accommodate small timing variations.

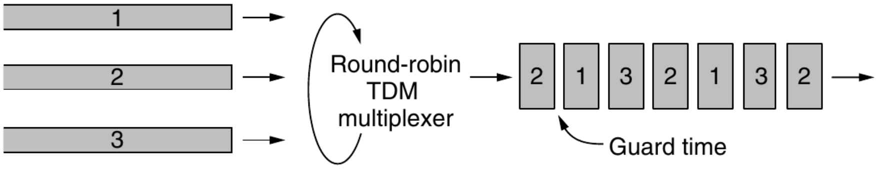


Figure 2-6: [Tanenbaum] TDM

CDM

There is a third kind of multiplexing that works in a completely different way than FDM and TDM. CDM (Code Division Multiplexing) is a form of spread spectrum communication in which a narrowband signal is spread out over a wider frequency band. This can make it more tolerant of interference, as well as allowing multiple signals from different users to share the same frequency band. Because code division multiplexing is mostly used for the latter purpose it is commonly called CDMA (Code Division Multiple Access). CDMA allows each station to transmit over the entire frequency spectrum all the time.

In CDMA, each bit time is subdivided into m short intervals called chips. Typically, there are 64 or 128 chips per bit, but in the example given here we will use 8 chips/bit for simplicity. Each station is assigned a unique m-bit code called a chip sequence. It is convenient to use a bipolar notation to write these codes as sequences of −1 and +1. We will show chip sequences in parentheses.

To transmit a 1 bit, a station sends its chip sequence. To transmit a 0 bit, it sends the negation of its chip sequence. No other patterns are permitted. Thus, for m = 8, if station A is assigned the chip sequence (−1 −1 −1 +1 +1 −1 +1 +1), it can send a 1 bit by transmiting the chip sequence and a 0 by transmitting (+1 +1 +1 −1 −1 +1 −1 −1).

Increasing the amount of information to be sent from b bits/sec to mb chips/sec for each station means that the bandwidth needed for CDMA is greater by a factor of m than the bandwidth needed for a station not using CDMA (assuming no changes in the modulation or encoding techniques). If we have a 1-MHz band available for 100 stations, with FDM each one would have 10 kHz and could send at 10 kbps (assuming 1 bit per Hz). With CDMA, each station uses the full 1 MHz, so the chip rate is 100 chips per bit to spread the station's bit rate of 10 kbps across the channel.

Each station has its own unique chip sequence. Let us use the symbol S to indicate the m -chip vector for station S , and $\overline{S}$ for its negation. All chip sequences are pairwise orthogonal, by which we mean that the normalized inner product of any two distinct chip sequences, **S** and **T** (written as $\mathbf{S} \bullet \mathbf{T}$), is 0. It is known how to generate such orthogonal chip sequences using a method known as Walsh codes. In mathematical terms, orthogonality of the chip sequences can be expressed as follows:

$\mathbf{S} \bullet \mathbf{T} \equiv \frac{1}{m} \sum_{i=1}^{m} S_i T_i = 0$

Note that if $\mathbf{S} \bullet \mathbf{T} = 0$, then $\mathbf{S} \bullet \overline{\mathbf{T}}$ is also 0. The normalized inner product of any chip sequence with itself is 1 :

$\mathbf{S} \bullet \mathbf{S} = \frac{1}{m} \sum_{i=1}^{m} S_i S_i = \frac{1}{m} \sum_{i=1}^{m} S_i^2 = \frac{1}{m} \sum_{i=1}^{m} (\pm 1)^2 = 1$

Also note that $\mathbf{S} \bullet \overline{\mathbf{S}} = -1$

The combined signal is

$$Y = \sum_i b_i \cdot S_i + (1 - b_i) \cdot \overline{S}_i$$

where $b_i \in \{0,1\}$. To recover station j' s signal, the receiver will just take the inner product of Y with S_j . i.e.,

$$\begin{aligned} Y \bullet S_j &= \sum_i b_i \cdot S_i \bullet S_j + (1 - b_i) \cdot \overline{S_i} \bullet S_j \\ &= b_j + \left(1 - b_j\right) \cdot -1 = 2b_j - 1 \end{aligned}$$

Equivalently, we have

$$b_j = \left(Y \bullet S_j + 1\right)/2$$

Link Layer

- We will refer to any device that runs a link-layer protocol as node.
- The communication channels that connect adjacent nodes along the communication path as links.
- In order to transfer data transferred from source node to destination node, it must be moved over each of the individual links in the end-to-end path.
- Transmitting nodes encapsulate datagram in a link-layer frame and transmits the frame into the link.

Link Layer Services

- **Framing**: A frame consists of a data field, in which the network-layer datagram is inserted, and a number of header fields.
- **Reliable delivery**: When a link-layer protocol provides reliable delivery service, it guarantees to move each network-layer datagram across the link without error.
- **Link access**: A medium access control (MAC) protocol specifies the rules by which a frame is transmitted onto the link. For point-to-point links, the MAC protocol is simple. The more interesting case is when multiple nodes share a single broadcast link—the so-called multiple access problem.
- **Error detection and correction**: The link-layer hardware in a receiving node can incorrectly decide that a bit in a frame is zero when it was transmitted as a one, and vice versa

Framing

- The bit stream received by the data link layer is not guaranteed to be error free.
- It is up to the data link layer to detect and, if necessary, correct errors.
- Data link layer to break up the bit stream into discrete frames, compute a short token called a checksum for each frame, and include the checksum in the frame when it is transmitted.
- When a frame arrives at the destination, the checksum is recomputed. If the newly computed checksum is different from the one contained in the

frame, the data link layer knows that an error has occurred and takes steps to deal with it.

- Four methods for Framing:
  - Byte count.
  - Flag bytes with byte stuffing.
  - Flag bytes with bit stuffing.
  - Physical layer coding violations.

Byte count

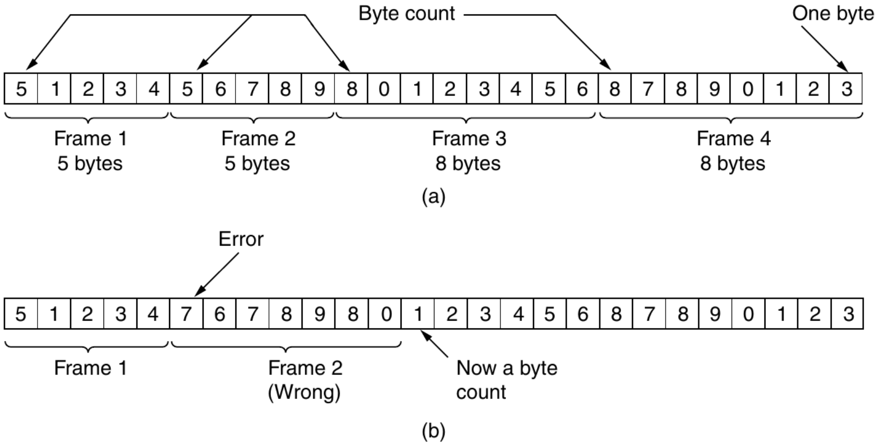


Figure [Tanenbaum] A byte stream (a) Without errors. (b) With one error

Flag bytes with byte stuffing

Two consecutive flag bytes indicate the end of one frame and the start of the next. Thus, if the receiver ever loses synchronization it can just search for two flag bytes to find the end of the current frame and the start of the next frame

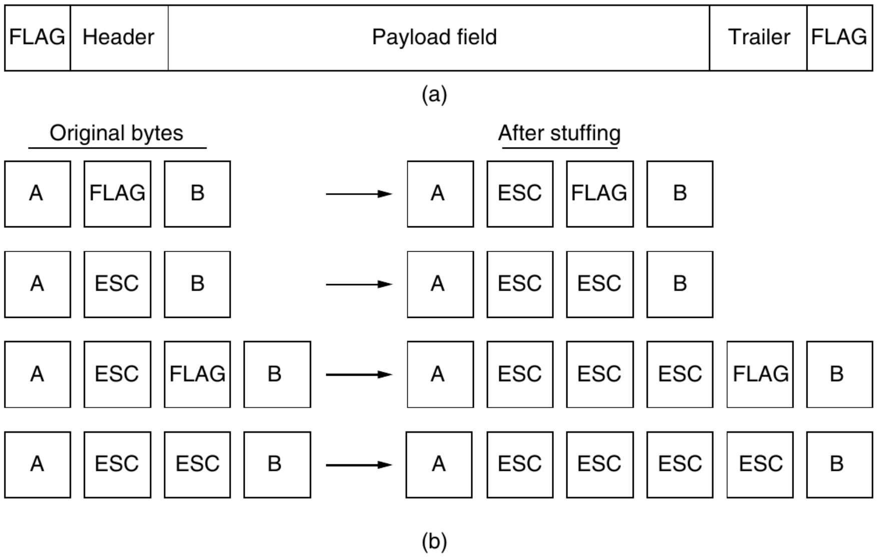


Figure [Tanenbaum] (a) A frame delimited by flag bytes. (b) Four examples of byte sequences before and after byte stuffing.

$2B$ overhead.

Flag bytes with bit stuffing

Each frame begins and ends with a special bit pattern, 01111110 or 0x7E in hexadecimal. This pattern is a flag byte.

Whenever the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a 0 bit into the outgoing bit stream.

When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically destuffs (i.e., deletes) the 0 bit.

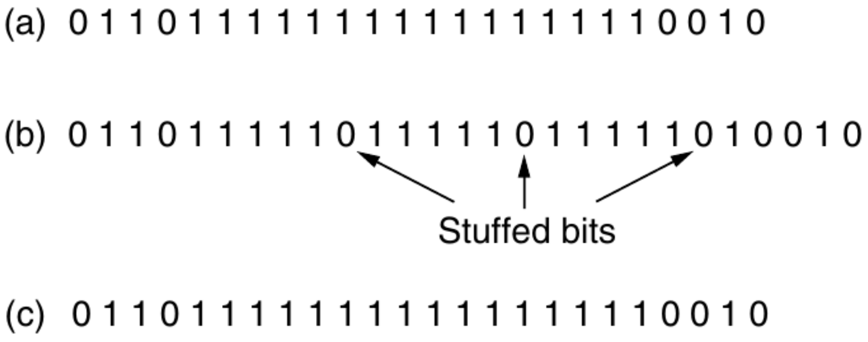

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0
(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0
Stuffed bits
(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Figure [Tanenbaum] (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in the receiver's memory after destuffing

$B/5$ overhead. (But hard disks prefer byte reads and writes)

Physical layer coding violations

- Encoding of bits as signals often includes redundancy to help the receiver.
- For example, in the 4B/5B line code 4 data bits are mapped to 5 signal bits to ensure sufficient bit transitions. We can use some reserved signals to indicate the start and end of frames.
- We are using "coding violations" to delimit frames.
- It is easy to find the start and end of frames and there is no need to stuff the data.

Error Control

Having solved the problem of marking the start and end of each frame, we come to the next problem: how to make sure all frames are eventually delivered to the network layer at the destination and in the proper order.

The usual way to ensure reliable delivery is to provide the sender with some feedback about what is happening at the other end of the line. Typically, the protocol calls for the receiver to send back special control frames bearing positive or negative acknowledgements about the incoming frames. If the sender receives a positive acknowledgement about a frame, it knows the frame has arrived safely. On the other hand, a negative acknowledgement means that something has gone wrong and the frame must be transmitted again.

An additional complication comes from the possibility that hardware troubles may cause a frame to vanish completely (e.g., in a noise burst). In this case, the receiver will not react at all, since it has no reason to react. Similarly, if the acknowledgement frame is lost, the sender will not know how to proceed. It should be clear that a protocol in which the sender transmits a frame and then waits for an acknowledgement, positive or negative, will hang forever if a frame is ever lost due to, for example, malfunctioning hardware or a faulty communication channel. This possibility is dealt with by introducing timers into the data link layer. When the sender transmits a frame, it generally also starts a timer. The timer is set to expire after an interval long enough for the frame to reach the destination, be processed there, and have the acknowledgement propagate back to the sender. Normally, the frame will be correctly received and the acknowledgement will get back before the timer runs out, in which case the timer will be canceled.

However, if either the frame or the acknowledgement is lost, the timer will go off, alerting the sender to a potential problem. The obvious solution is to just transmit the frame again. However, when frames may be transmitted multiple times there is a danger that the receiver will accept the same frame two or more times and pass it to the network layer more than once. To prevent this from happening, it is generally necessary to assign sequence numbers to outgoing frames, so that the receiver can distinguish retransmissions from originals.

Flow Control

Another important design issue that occurs in the data link layer (and higher layers as well) is what to do with a sender that systematically wants to transmit frames faster than the receiver can accept them. This situation can occur when the sender is running on a fast, powerful computer and the receiver is running on a slow, low-end machine.

Two approaches are commonly used. In the first one, **feedback-based flow control**, the receiver sends back information to the sender giving it permission to send more data, or at least telling the sender how the receiver is doing. In the second one, **rate-based flow control**, the protocol has a built-in mechanism that limits the rate at which senders may transmit data, without using feedback from the receiver.

Error detection and correction

- Transmission errors are unavoidable. We need to develop techniques to deal with them.
- One strategy is to include enough redundant information to enable the receiver to deduce what the transmitted data must have been — Forward Error Correction (FEC)
- The other is to include only enough redundancy to allow the receiver to deduce that an error has occurred and have it request a retransmission.
- When noise is unavoidable like in wireless communication, FEC is must and when there is low probability of error then detection is better.
- A frame consists of m data (i.e., message) bits and r redundant (i.e. check) bits.
- Block code: the r check bits are computed solely as a function of the m data bits with which they are associated
- Systematic code: the m data bits are sent directly, along with the check bits, rather than being encoded themselves before they are sent.
- Linear code: the r check bits are computed as a linear function of the m data bits. Exclusive OR (XOR) or modulo 2 addition is a popular choice.

- The codes we will look at are linear, systematic block codes unless otherwise noted.
- Let the total length of a block be n (i.e., $n = m+r$). We will describe this as an (n, m) code. An n-bit unit containing data and check bits is referred to as an n-bit codeword. The code rate is the fraction of the codeword that carries non-redundant information i.e., m/n.

Hamming Codes

- Given two code words 10001001 and 10110001 — is to possible to determine how many corresponding bits differ?
- This difference is called the Hamming distance
- If two code words are a Hamming distance of d apart, d single-bit errors are needed to convert one into the other.
- In most data transmission applications, all 2^m possible data messages are legal, but due to the way the check bits are computed, not all of the 2^n possible codewords are used. In fact, when there are r check bits, only the small fraction of $2^m/2^n$ or $1/2^r$ of the possible messages will be legal codewords.
- Given the algorithm for computing the check bits, it is possible to construct a complete list of the legal codewords, and from this list to find the two codewords with the smallest Hamming distance. This distance is the Hamming distance of the complete code.
- To reliably detect d errors, you need a distance $d + 1$ code because with such a code there is no way that d single-bit errors can change a valid codeword into another valid codeword. When the receiver sees an illegal codeword, it can tell that a transmission error has occurred.
- Similarly, to correct d errors, you need a distance $2d+1$ code because that way the legal codewords are so far apart that even with d changes the original codeword is still closer than any other codeword. This means the original codeword can be uniquely determined based on the assumption that a larger number of errors are less likely.
- We want to design a code with m message bits and r check bits that will allow all single errors to be corrected.
- Each of the 2^m legal messages has n illegal codewords at a distance of 1 from it.
- Each of the 2^m legal messages requires $n + 1$ bit patterns dedicated to it.
- We get the requirement that $2^m(n + 1) \leq 2^n \rightarrow (m + r + 1) \leq 2^r$.
- To correct 2 bit errors $\rightarrow 2^m(\binom{n}{2} + 1) \leq 2^n$
- (For 1 bit error correction, consider $|m_i| = 7$) Place check bits at index power of two because then $d(C(m_1), C(m_2)) \geq 3$ whenever $d(m_1, m_2) \geq 1$ as index of bit difference will not be a power of 2.

Convolutional Code

- Not a block code.
- In a convolutional code, an encoder processes a sequence of input bits and generates a sequence of output bits. There is no natural message size or encoding boundary as in a block code.
- The output depends on the current and previous input bits. That is, the encoder has memory. The number of previous bits on which the output depends is called the **constraint length** (includes current bit) of the code.
- Convolutional codes are specified in terms of their rate ($\frac{\text{\# Input bits}}{\text{\# Output Bits produced}}$) and constraint length.

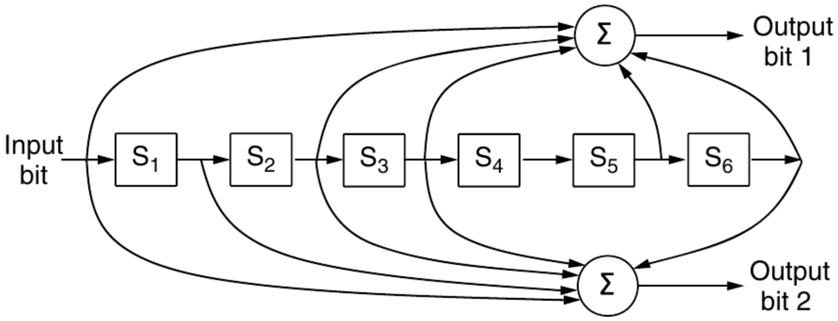


Figure: [Tanenbaum] The NASA binary convolutional code used in 802.11.

In figure, each input bit on the left-hand side produces two output bits on the right-hand side that are XOR sums of the input and internal state. Since it deals with bits and performs linear operations, this is a binary, linear convolutional code. Since 1 input bit produces 2 output bits, the code rate is 1/2. It is not systematic since none of the output bits is simply the input bit.

The internal state is kept in six memory registers. Each time another bit is input the values in the registers are shifted to the right. For example, if 111 is input and the initial state is all zeros, the internal state, written left to right, will become 100000, 110000, and 111000 after the first, second, and third bits have been input. The output bits will be 11, followed by 10, and then 01. It takes seven shifts to flush an input completely so that it does not affect the output. The constraint length of this code is thus k = 7.

A convolutional code is decoded by finding the sequence of input bits that is most likely to have produced the observed sequence of output bits (which includes any errors). For small values of k, this is done with a widely used algorithm developed by Viterbi (Forney, 1973). The algorithm walks the observed sequence, keeping for each step and for each possible internal state the input sequence that would have produced the observed sequence with the fewest errors. The input sequence requiring the fewest errors at the end is the most likely message. Convolutional codes have been popular in practice because it is easy to factor the uncertainty of a bit being a 0 or a 1 into the decoding.

Reed-Solomon Code

- Are linear block codes, and they are often systematic too.
- Unlike Hamming codes, which operate on individual bits, Reed-Solomon codes operate on m bit symbols.
- Reed-Solomon codes are based on the fact that every n degree polynomial is uniquely determined by n + 1 points. For example, a line having the form ax + b is determined by two points. Extra points on the same line are redundant, which is helpful for error correction. Imagine that we have two data points that represent a line and we send those two data points plus two check points chosen to lie on the same line. If one of the points is received in error, we can still recover the data points by fitting a line to the received points. Three of the points will lie on the line, and one point, the one in error, will not. By finding the line we have corrected the error.
- Reed-Solomon codes are actually defined as polynomials that operate over finite fields, but they work in a similar manner. For m bit symbols, the codewords are $2^m - 1$ symbols long. A popular choice is to make m = 8 so that symbols are bytes. A codeword is then 255 bytes long. The (255, 233) code is widely used; it adds 32 redundant symbols to 233 data symbols. Decoding with error correction is done with an algorithm developed by Berlekamp and Massey that can efficiently perform the fitting task for moderate-length codes
- Reed-Solomon codes are widely used in practice because of their strong error-correction properties, particularly for burst errors. Because they are based on m bit symbols, a single-bit error and an m-bit burst error are both treated simply as one symbol error.
- When $2t$ redundant symbols are added, a Reed-Solomon code is able to correct up to t errors in any of the transmitted symbols. This means, for example, that the (255, 233) code, which has 32 redundant symbols, can correct up to 16 symbol errors. Since the symbols may be consecutive and they are each 8 bits, an error burst of up to 128 bits can be corrected.

Error Detecting Codes

We will examine three different error-detecting codes. They are all linear, systematic block codes.

To see how they can be more efficient than error-correcting codes, consider:-

Parity

- Consider a case where a single parity bit is appended to the data.
- The parity bit is chosen so that the number of 1 bits in the codeword is even (or odd). For example, when 1011010 is sent in even parity, a bit is

added to the end to make it 10110100. With odd parity 1011010 becomes 10110101.

- A code with a single parity bit has a distance of 2, since any single-bit error produces a codeword with the wrong parity. This means that it can detect single-bit errors.
- One difficulty with this scheme is that a single parity bit can only reliably detect a single-bit error in the block. If the block is badly garbled by a long burst error, the probability that the error will be detected is only 0.5.
- The odds can be improved considerably if each block to be sent is regarded as a rectangular matrix *n* bits wide and *k* bits high. Now, if we compute and send one parity bit for each row, up to *k* bit errors will be reliably detected as long as there is at most one error per row. However, there is something else we can do that provides better protection against burst errors: we can compute the parity bits over the data in a different order than the order in which the data bits are transmitted. Doing so is called interleaving. In this case, we will compute a parity bit for each of the n columns and send all the data bits as k rows, sending the rows from top to bottom and the bits in each row from left to right in the usual manner. At the last row, we send the n parity bits. This transmission order is shown in below Fig for n = 7 and k = 7.

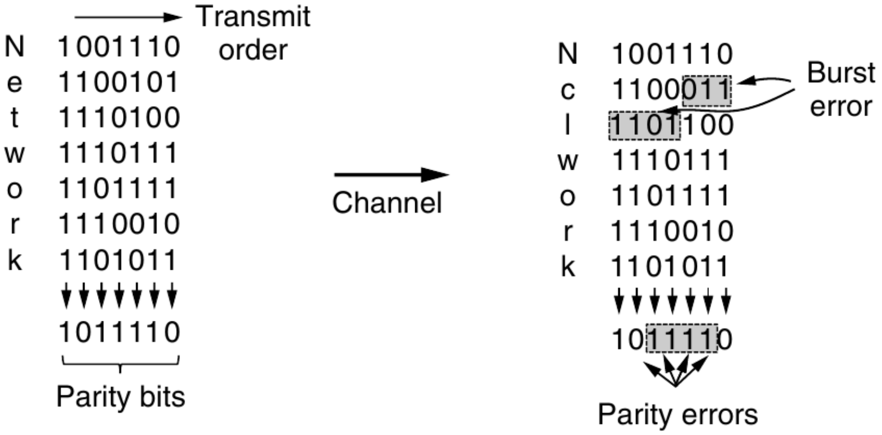


Figure: [Tanenbaum] Interleaving of parity bits to detect a burst error. (A burst error does not imply that all the bits are wrong; it just implies that at least the first and last are wrong.)

Interleaving is a general technique to convert a code that detects (or corrects) isolated errors into a code that detects (or corrects) burst errors. This method uses *n* parity bits on blocks of kn data bits to detect a single burst error of length n or less.

A burst of length $n+1$ will pass undetected, however, if the first bit is inverted, the last bit is inverted, and all the other bits are correct. If the block is badly garbled by a long burst or by multiple shorter bursts, the probability that any of the n columns will have the correct parity by accident is 0.5, so the probability of a bad block being accepted when it should not be is 2^{-n} .

Checksum

Checksums are usually based on a running sum of the data bits of the message. The checksum is usually placed at the end of the message, as the complement of the sum function. This way, errors may be detected by summing the entire received codeword, both data bits and checksum. If the result comes out to be zero, no error has been detected.

One example of a checksum is the 16-bit Internet checksum used on all Internet packets as part of the IP protocol (Braden et al., 1988). This checksum is a sum of the message bits divided into 16-bit words. Because this method operates on words rather than on bits, as in parity, errors that leave the parity unchanged can still alter the sum and be detected. For example, if the lowest order bit in two different words is flipped from a 0 to a 1, a parity check across these bits would fail to detect an error. However, two 1s will be added to the 16-bit checksum to produce a different result. The error can then be detected.

Cyclic Redundancy Check (CRC)

- Aka Polynomial codes are based upon treating bit strings as representations of polynomials with coefficients of 0 and 1 only. A k-bit frame is regarded as the coefficient list for a polynomial with k terms, ranging from x^{k-1} to x^0. The high-order (leftmost) bit is the coefficient of x^{k-1} and so on.
- Polynomial arithmetic is done modulo 2, according to the rules of algebraic field theory. It does not have carries for addition or borrows for subtraction. Both addition and subtraction are identical to exclusive OR.
- Long division is carried out in exactly the same way as it is in binary except that the subtraction is again done modulo 2.
- When the polynomial code method is employed, the sender and receiver must agree upon a generator polynomial, $G(x)$, in advance. Both the high- and low- order bits of the generator must be 1. To compute the CRC for some frame with m bits corresponding to the polynomial $M(x)$, the frame must be longer than the generator polynomial. The idea is to append a CRC to the end of the frame in such a way that the polynomial represented by the checksummed frame is divisible by $G(x)$. When the receiver gets the checksummed frame, it tries dividing it by $G(x)$. If there is a remainder, there has been a transmission error.
- The algorithm for computing the CRC is as follows:

1. Let r be the degree of $G(x)$. Append r zero bits to the low-order end of the frame so it now contains $m + r$ bits and corresponds to the polynomial $x^r M(x)$.
2. Divide the bit string corresponding to $G(x)$ into the bit string coresrponding to $x^r M(x)$, using modulo 2 division.
3. Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to $x^r M(x)$ using modulo 2 subtraction. The result is the checksummed frame to be transmitted. Call its polynomial $T(x)$.

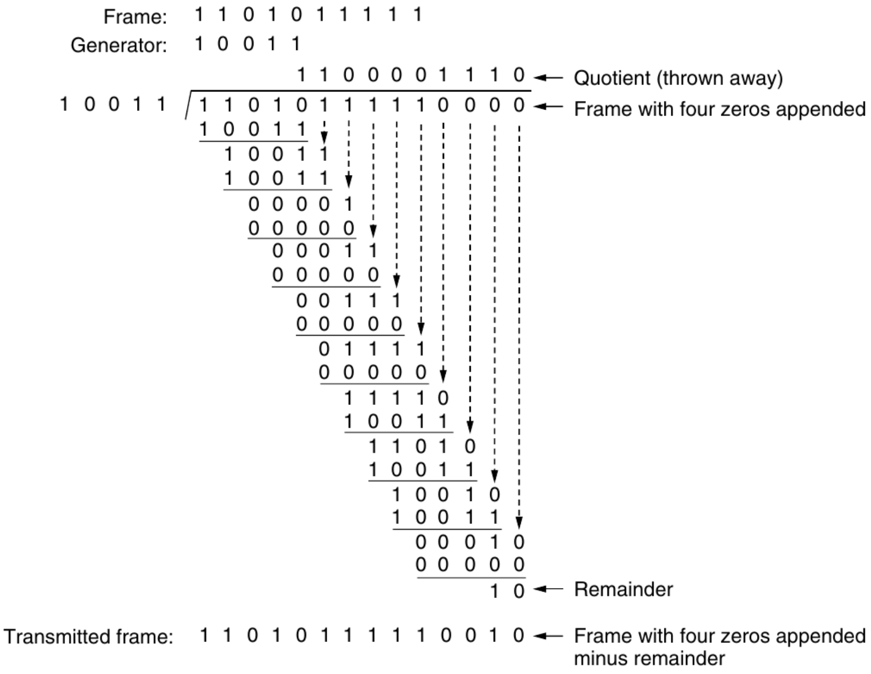


Figure: [Tanenbaum] Example Calculation of CRC